Greenfoot spiekbrief

Wezens, spelers, acteurs, . . .

Je moet de klasse *Actor* (indirect) uitbreiden en de methode *act( )* implementeren. Deze methode wordt elke tijdspuls één keer uitgevoerd.

```
public class Auto extends Actor {

    public void act () {
        // wordt elke tijdspuls herhaald
        ...
    }

}
```

Bewegen

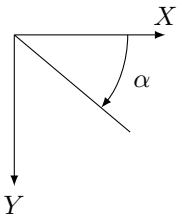
Relatieve beweging:

move (afstand)	Beweeg in de huidige richting over een *afstand* in roostereenheden
turn (graden)	Draai (ter plaatse) over een hoek in *graden*

Absolute beweging:

setLocation (x,y)	Spring rechtstreeks naar een roosterpunt
setRotation (hoek)	Keer u (ter plaatse) naar een richting
turnTowards (x,y)	Keer u (ter plaatse) naar een roosterpunt

Coördinaten:



naar rechts	0
naar onder	90
naar links	180
naar boven	270

Roosterpunt (0,0) = linkerbovenhoek v/d wereld.

Opvragen:

getX ( )	Huidige X-coördinaat in het wereldrooster
getY ( )	Huidige Y-coördinaat in het wereldrooster
getRotation ( )	Huidige (kijk)richting
isAtEdge ( )	Is de acteur aan de rand van de wereld?

(Methoden *getX*, *getY* en *isAtEdge* gooien een uitzondering op als de acteur zich niet op de wereld bevindt.)

Imports

Om de greenfootbibliotheek te gebruiken, moet je de greenfootklassen bovenaan importeren.

```
import greenfoot.*;
```

Afbeeldingen

Afbeelding waarmee een acteur getoond wordt:

getImage ( )	Huidige afbeelding (kan *null* zijn)
setImage (afb)	Stel nieuwe afbeelding in (*null* = onzichtbaar)
setImage (naam)	Idem, maar met *naam* v/h afbeeldingenbestand

Afbeeldingen zijn van het type *GreenfootImage*.

```
GreenfootImage image =
    new GreenfootImage ("vlag.png");
...
setImage (image);
```

Type

Soms moet je niet een acteur zelf opgeven als parameter (object) maar het *type* van een object (klasse). Gebruik hiervoor de '`.class`'-notatie.

```
// alle beren op de wereld
return
  getWorld().getObjects(Beer.class);
```

Objecten van een deelklasse tellen ook mee.

Overlappen e.d.

intersects (andere)
: Overlapt deze acteur met de *andere* acteur?

isTouching (type)
: Overlapt deze acteur met een acteur van dit *type*?

removeTouching (type)
: Verwijder (hoogstens) één overlappende acteur van dit *type*.

getIntersectingObjects (type)
: Geef alle overlappende acteurs terug van dit *type*.

getOneIntersectingObject (type)
: Geef (hoogstens) één overlappende acteur terug van dit *type*.

getObjectsInRange(straal,type)
: Geef alle acteurs terug van dit *type* binnen de gegeven *straal*.

Overlappen e.d. (vervolg)

getNeighbours(afstand,`false`*,type)*
: Geef alle acteurs terug van dit *type* binnen de gegeven *afstand*, gerekend in aantal horizontale/verticale stappen in het puntenrooster.

getNeighbours(afstand,`true`*,type)*
: Idem, maar gerekend in aantal horizontale/verticale of *diagonale* stappen.

getObjectsAtOffset (dx,dy,type)
: Geef alle acteurs terug van dit *type* die een gegeven punt overlappen, in relatieve coördinaten.

getOneObjectAtOffset (dx,dy,type)
: Idem, maar geeft hoogstens één dergelijke acteur terug.

De wereld

De wereld is een object van een klasse die *World* uitbreidt. Bij constructie geef je de dimensies op van het puntenrooster en de grootte van de cellen. Bijv. schaakbord met cellen van 50×50 pixels:

```
public class Schaakbord extends World {
  public Schaakbord () {
    super (8, 8, 50);
  }
  ...
}
```

Bijv. 80000 cellen van 2×2 pixels, 200 rijen, 400 kolommen (= 400 cellen breed, 200 cellen hoog):

```
    super (400, 200, 2, false);
```

Vierde argument `false` laat toe dat acteurs zich *buiten* de wereld begeven.

Ook *World* heeft een methode *act( )* die elke tijdspuls één keer uitgevoerd worden.

Acteurs op de wereld

addObject (acteur,x,y)	Voeg de *acteur* toe op een roosterpunt
removeObject (acteur)	Verwijder de *acteur*
removeObjects (lijst)	Verwijder alle acteurs in de *lijst*
getObjects (type)	Geef alle acteurs uit de wereld terug van een bepaald *type*
getObjectsAt (x,y,type)	Idem, maar moeten ook het roosterpunt overlappen. (In absolute coördinaten.)

Een acteur kan zijn wereld opvragen met de methode *getWorld ( )*.

Voorbeeld: verwijder jezelf van de wereld

```
getWorld().removeObject(this);
```

Er bestaat ook een variant die een wereld van het juiste type teruggeeft:

```
MijnWereld wereld =
    getWorldOfType(MijnWereld.class);
```

Toetsenbord

(Klassenmethoden uit de klasse *Greenfoot*.)

Greenfoot.isKeyDown (toets)	Is de *toets* op dit moment ingedrukt?
Greenfoot.getKey ( )	De laatste ingedrukte toets (sinds de vorige *getKey*)

Toetsen worden voorgesteld als strings:

Lettertoetsen	`a b c ...`
Cijfertoetsen	`0 1 2 ...`
Leestekens	`. ? ! , ; : ...`
Pijltjes	`right down left up`
Functietoetsen	`F1 F2 ... F12`
Andere	`enter space tab escape backspace`
'Dode' toetsen	`shift control`

(De methode *getKey ( )* geeft ook hoofdletters terug.)

```
// gebruik de pijltjestoetsen
// om van richting te veranderen
if (Greenfoot.isKeyDown("left")) {
  turn (270);
} else if (Greenfoot.isKeyDown("right")) {
  turn (90);
} else if (Greenfoot.isKeyDown("down")) {
  turn (180);
}
move (1);
```

Diversen

(Klassenmethoden uit de klasse *Greenfoot*.)

Greenfoot.getRandomNumber(n)
: 'Willekeurig' geheel getal in het bereik [0,*n*-1]

Greenfoot.playSound(bestandsnaam)
: Speel een geluid (*naam* = bestand)

Volgorde

(Methoden van *World*)

setPaintOrder(Klasse1.class, Klasse2.class, ...)
: Volgorde waarin objecten worden getekend. Objecten van *Klasse1* worden *boven* die van *Klasse2* getekend, ...

setActOrder(Klasse1.class, Klasse2.class, ...)
: Volgorde waarin de '*act ()*'s worden opgeroepen. Eerst bij objecten van *Klasse1*, ...

Simulatie

(Klassenmethoden uit de klasse *Greenfoot*.)

Greenfoot.stop ( )	Leg de simulatie stil
Greenfoot.setSpeed (s)	Stel de snelheid v/d simulatie in (*s* tussen 1 en 100)

Belangrijk.
Er staat nog meer informatie in de elektronische documentatie (API).